



面向对象的软件构造实践

实验二
(4学时)

2024春



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

用户界面

事件处理

图形系统

数据存储
与展示

音乐音效

网络编程

模块功能：完成静态页面的设计

① 完成游戏首页的
界面设计



② 完成游戏难度选择
页面的界面设计



用户界面

事件处理

图形系统

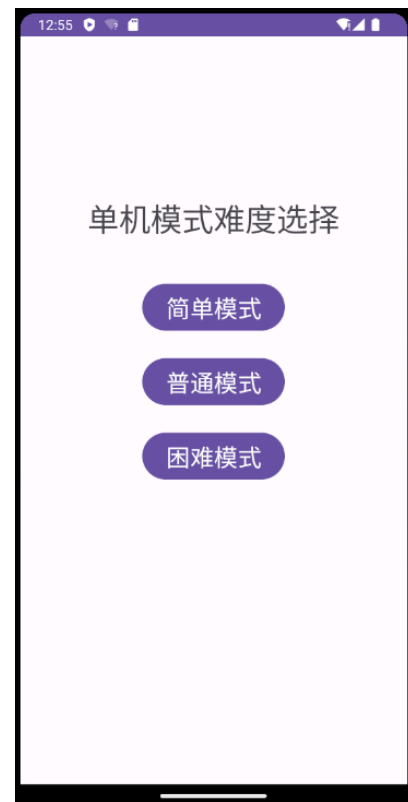
数据存储
与展示

音乐音效

网络编程

模块功能：完成页面切换和数据传递

- ① 使用事件监听机制，完成从游戏首页到难度选择页面的跳转和数据传递



- 掌握TextView, Button等常见**视图控件**的使用;
- 了解常见的Android布局, 掌握**约束布局**的使用;
- 理解Android事件处理机制, 掌握**基于监听的事件处理**的实现;
- 初识Intent, 掌握使用Intent进行**页面切换和数据传递**的方式;
- 理解Activity的生命周期。

2.1 Android用户界面

2.2 Android视图控件

2.3 Android布局管理

2.4 Android生命周期

2.5 Android事件处理机制

2.6 Android页面切换与数据传递

2.1 Android用户界面

- Android用户界面(User Interface,UI)是人与手机之间数据传递、交互信息的重点对话接口，是软件用户友好性的重要体现。

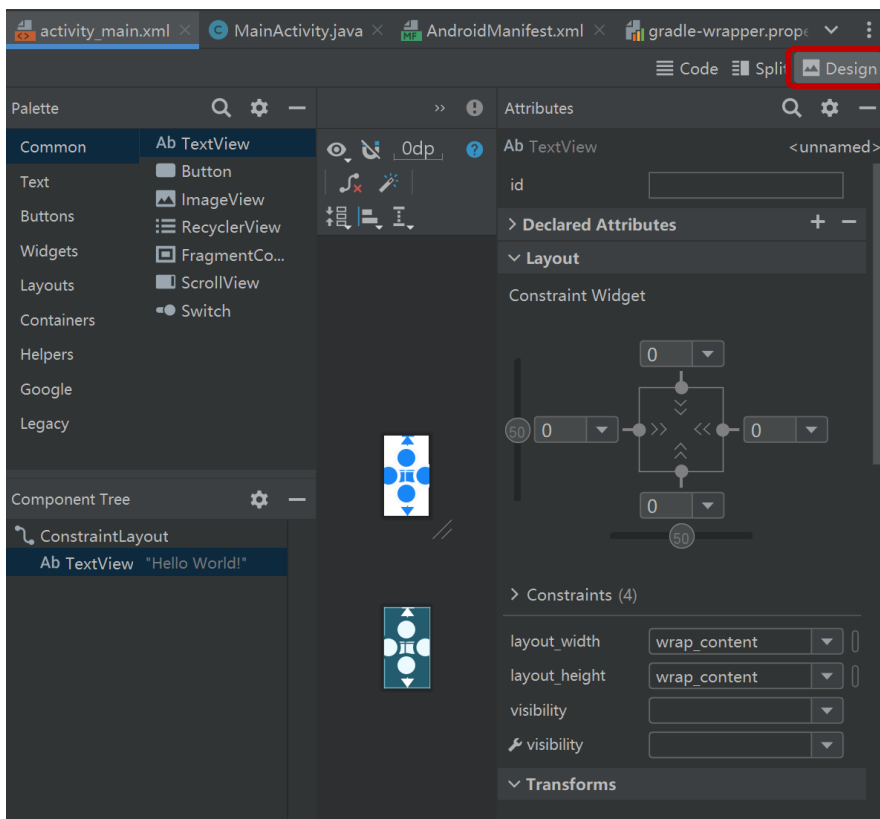


用户界面中可显示的内容有很多，如文本框、按钮、列表框、图片、进度条等，这些用户界面元素被称为**控件 View**。大部分控件是可见的。在Android中，所有的可视控件都继承自View类。

2.1 Android用户界面

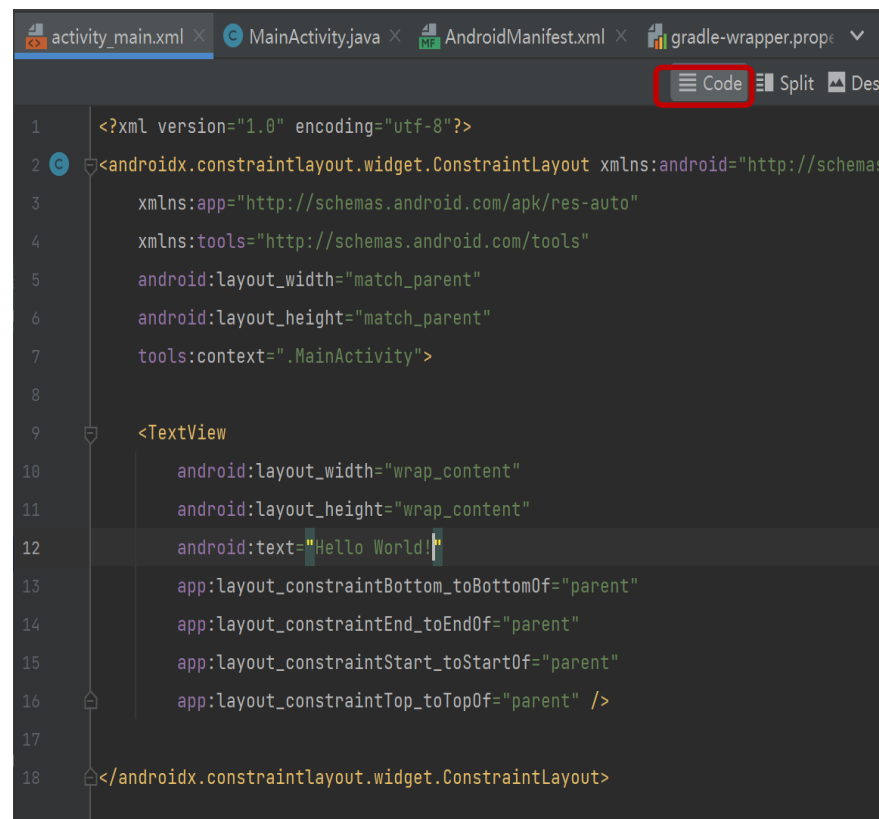
如何编写用户界面

使用**Design**模式打开xml布局文件，
在可视化编辑器中拖放控件，设置属性。



OR

在**Code**模式下打开xml布局文件，编写
xml代码来操作控件 **(推荐使用)**。



2.2 Android视图控件

- View对象用于在屏幕上绘制可供用户交互的内容，是一个数据体，它的**属性**决定了此控件在页面上如何显示，并担任**交互事件接受者**的角色。
- 对View类及其子类的属性进行设置，可以在布局文件XML中设置，也可以通过成员方法在Java代码文件中动态设置。

2.2 Android视图控件

View类的常用属性与方法

属 性	对 应Java 方法	说 明
android:background	setBackgroundColor(int color)	设置背景颜色
android:id	setId(int)	为组件设置可通过findViewById方法获取的标识符
	findViewById(int id)	与id所对应的组件建立关联
android:alpha	setAlpha(float)	设置透明度，取值[0, 1]之间
android:visibility	setVisibility(int)	设置组件的可见性
android:clickable	setClickable(boolean)	设置组件是否响应单击事件

2.2 Android视图控件

TextView

用来在界面上显示一段文本信息。

元素属性	说明
android:id	文本标签标识
android:layout_width	指定TextView的宽度高度，通常取值： 1. match_parent : 控件的大小和父元素的大小一样； 2. wrap_content : 由控件内容决定当前控件的大小； 3. 以像素为单位的固定值：不同手机可能会出现适配问题。
android:layout_height	
android:text	指定TextView的文本内容
android:textSize	TextView的字体大小，以sp为单位
android:textColor	指定文字的颜色
android:gravity	文字的对齐方向：top, bottom, left, right, center

2.2 Android视图控件

TextView

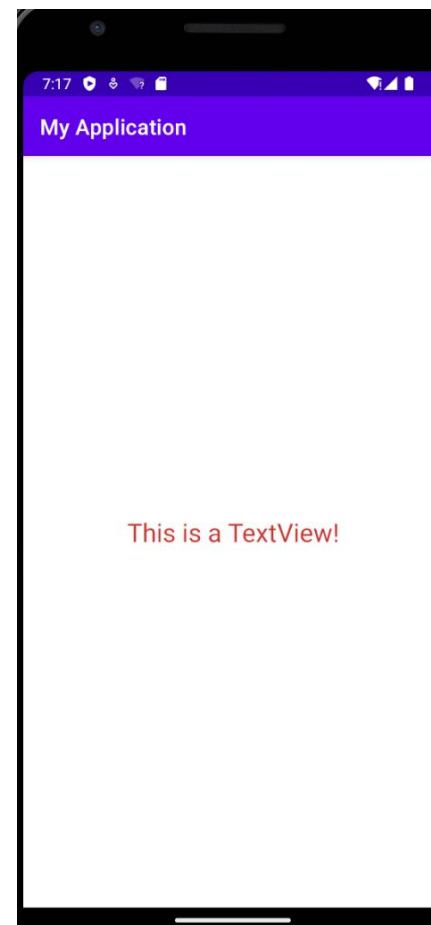
在页面中增加一个TextView。

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a TextView!"
    android:id="@+id/textView"
    android:textColor="#CF352E"
    android:textSize="24sp"
```

注意id的设置方法

控件设置id属性后，可以在Java代码中通过
findViewById的方法找到界面上的视图组件。


```
TextView txt = findViewById(R.id.textView);
```



2.2 Android视图控件

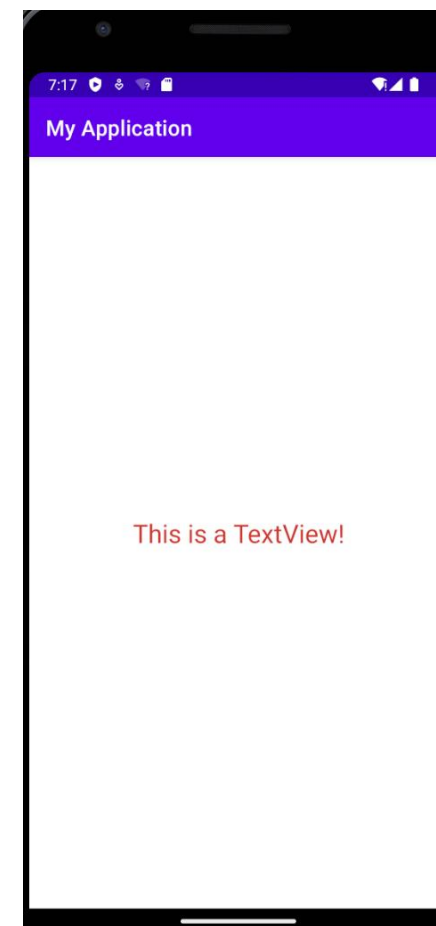
TextView

可以将要显示的文字放到 **res/values/strings.xml** (此文件系统默认创建)

```
<resources>
  <string name="app_name">My Application</string>
   <string name="str">This is a TextView!</string>
</resources>
```

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/str"
  android:id="@+id/textView"
  android:textColor="#CF352E"
  android:textSize="24sp"
```

注意strings.xml的使用方法



2.2 Android视图控件

Button

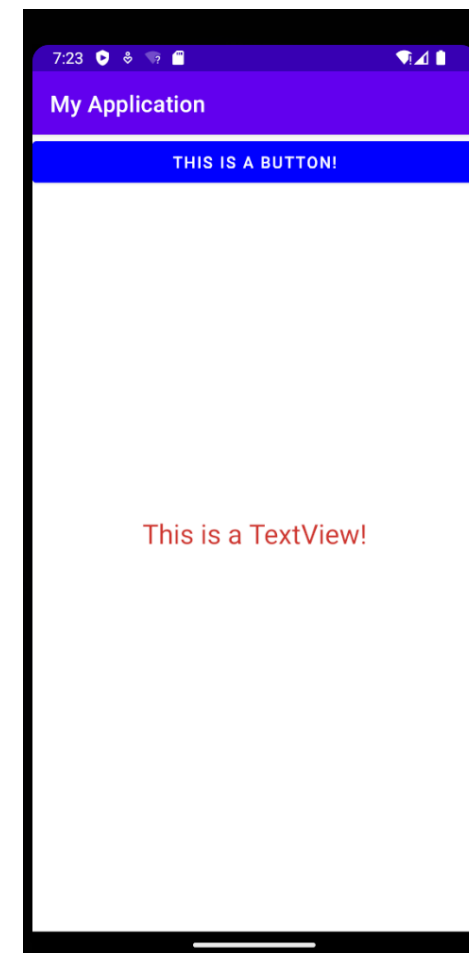
用来和用户交互。

添加一个Button控件。

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="This is a Button!"
    android:backgroundTint="@color/blue"/>
```

backgroundTint属性改变背景颜色，
blue定义在res/values/colors.xml中。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="blue">#0000FF</color>
</resources>
```



RadioGroup与RadioButton

单选组件RadioGroup用于多项选择中只允许任选其中一项的情形。
单选组件RadioGroup由一组单选按钮RadioButton组成。

方 法	功 能
<code>isChecked()</code>	判断选项是否被选中
<code>getText()</code>	获取单选按钮的文本内容

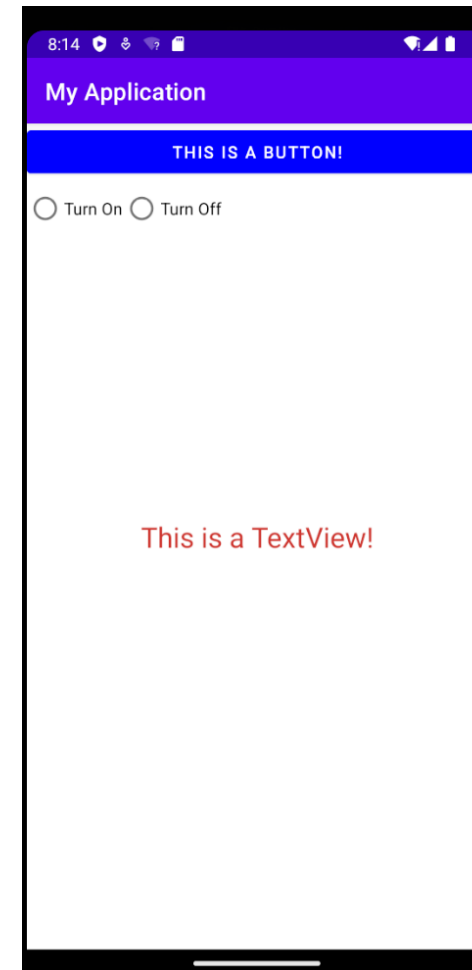
2.2 Android视图控件

RadioGroup与RadioButton

在页面上增加两个单选按钮，开启和关闭。

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintTop_toBottomOf="@id/btn">
    <RadioButton android:id="@+id/radio_on"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Turn On"/>
    <RadioButton android:id="@+id/radio_off"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Turn Off"/>
```

单选按钮的排列方向



2.2 Android视图控件

ImageView

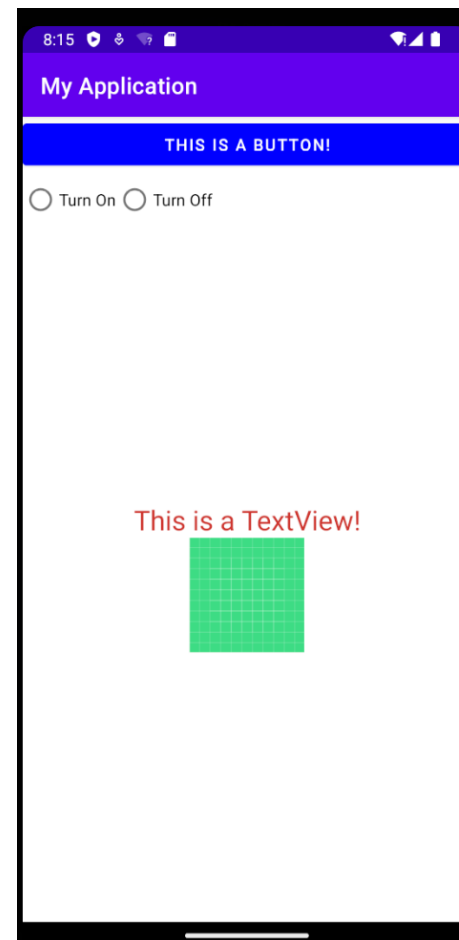
用来在页面上展示图片。

添加一个ImageView控件显示图片，图片放在res/drawable文件夹下，通过src属性设置要展示的图片。

```
<ImageView
    android:id="@+id/image"
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:src="@drawable/ic_launcher_background"
```

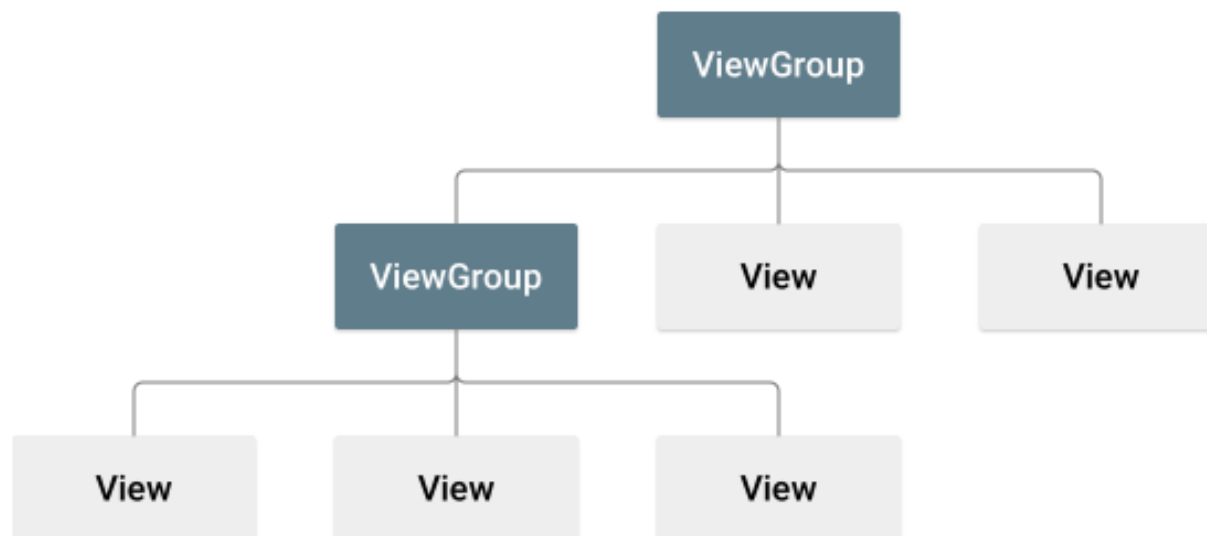
思考：如何控制各个视图控件的排列方式？
控件xml代码应该添加到哪里？

Android布局管理



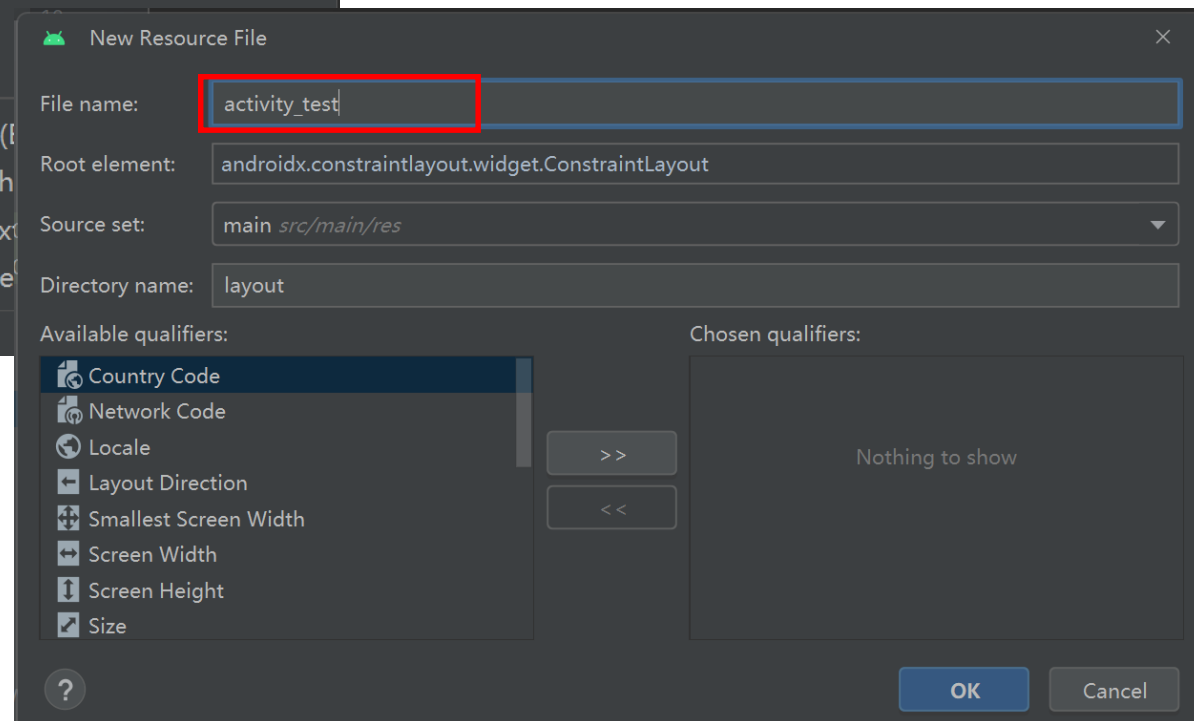
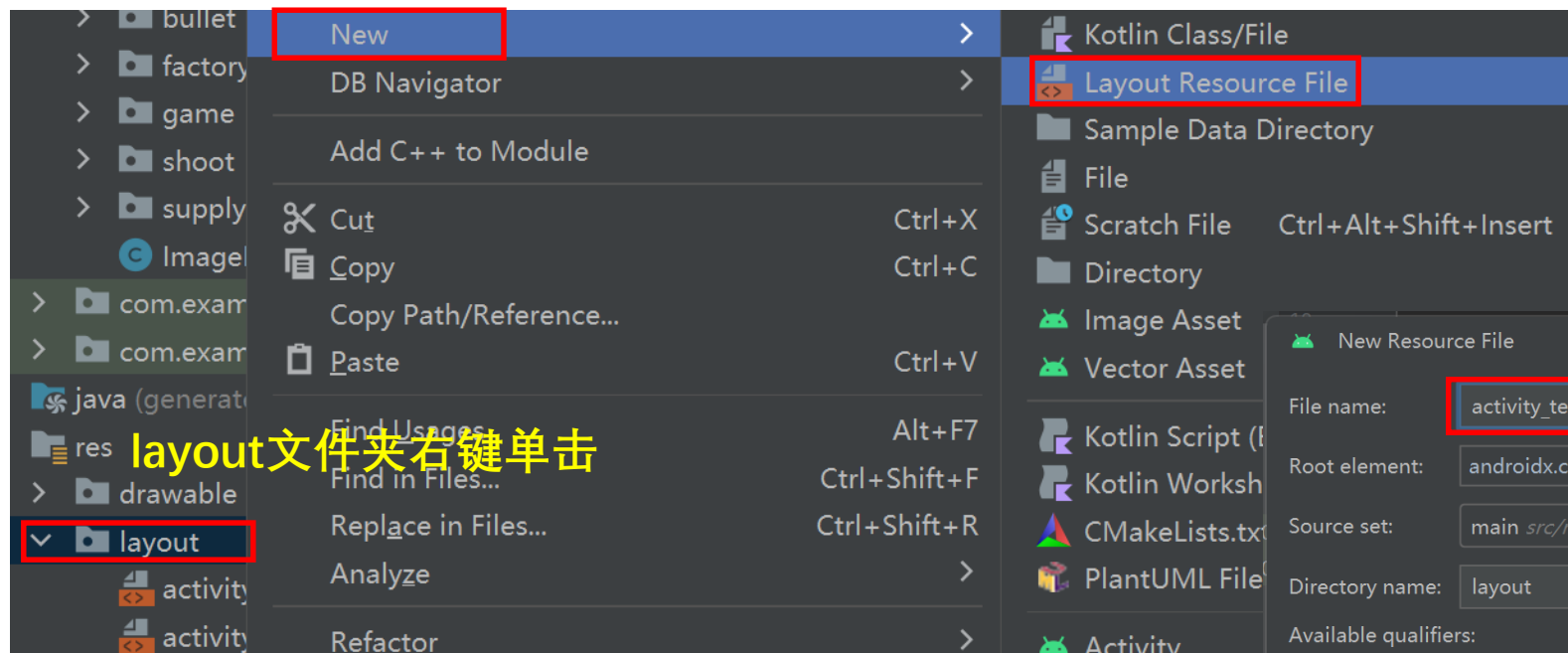
2.3 Android布局管理

- Android系统的布局管理指的是在XML布局文件中设置组件的大小、间距、排列及对齐方式等。
- Android布局中的所有元素均使用 **View** 和 **ViewGroup** 对象的层次结构进行构建。View 通常用于绘制用户可看到并与之交互的内容（Button, TextView等）。ViewGroup 则是不可见的容器，用于定义 View 和其他 ViewGroup对象的布局结构。
- Android布局管理，通过多层布局的嵌套，完成一些比较复杂页面的开发。



2.3 Android布局管理

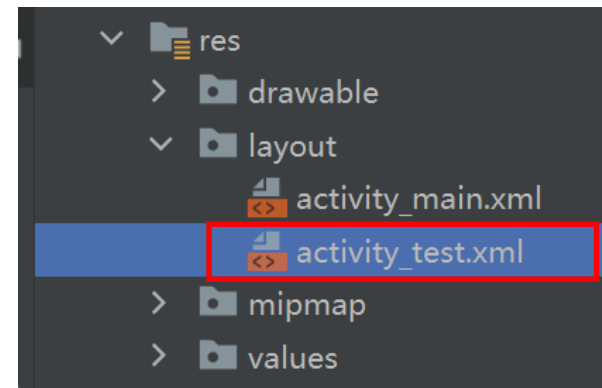
创建布局文件



2.3 Android布局管理

编写XML

- 创建成功的布局文件位于res/layout文件夹下;
- 布局文件都必须只含一个根元素, 且该元素必须是视图对象或ViewGroup对象;
- 定义根元素后, 以子元素的形式添加其他布局对象, 从而逐步构建定义布局的视图层次结构。



```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button android:id="@+id/btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a Button!"
        android:backgroundTint="@color/blue"
        android:layout_marginLeft="50dp"
        android:layout_marginRight="50dp"
        app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

加载XML

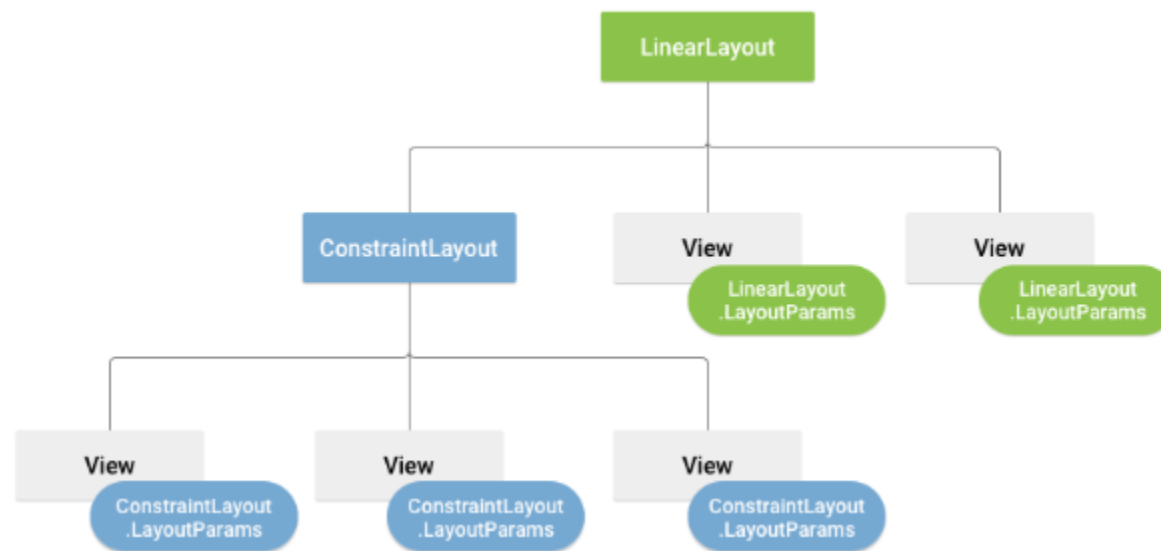
- 编译应用时，系统会将每个 XML 布局文件编译成View资源；
- 在 `Activity.onCreate()` 回调实现中调用 `setContentView()`，并以 `R.layout.layout_file_name` 形式向应用代码传递对布局资源的引用；
- 例如，如果 XML 布局保存为 `main_layout.xml`，应通过如下方式为 Activity 加载布局资源：

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main_layout);  
}
```

- 启动Activity时，Android 框架会调用 Activity 中的 `onCreate()` 回调方法，从而加载应用代码中的布局资源；

常见布局

- 每种布局Layout都有自己的布局参数，称为LayoutParams，父视图使用布局参数为子视图定义尺寸和位置。注意，这些参数要在**子视图控件**中进行配置。



实际开发时根据**需求**选择布局

常见布局

- 帧布局FrameLayout

只显示一个View对象，显示对象会固定在屏幕的左上角，不能指定位置；有多个对象时，后一个会覆盖前一个。



- 绝对布局AbsoluteLayout

以坐标的方式，指定View对象的具体位置，左上角坐标为 (0,0)

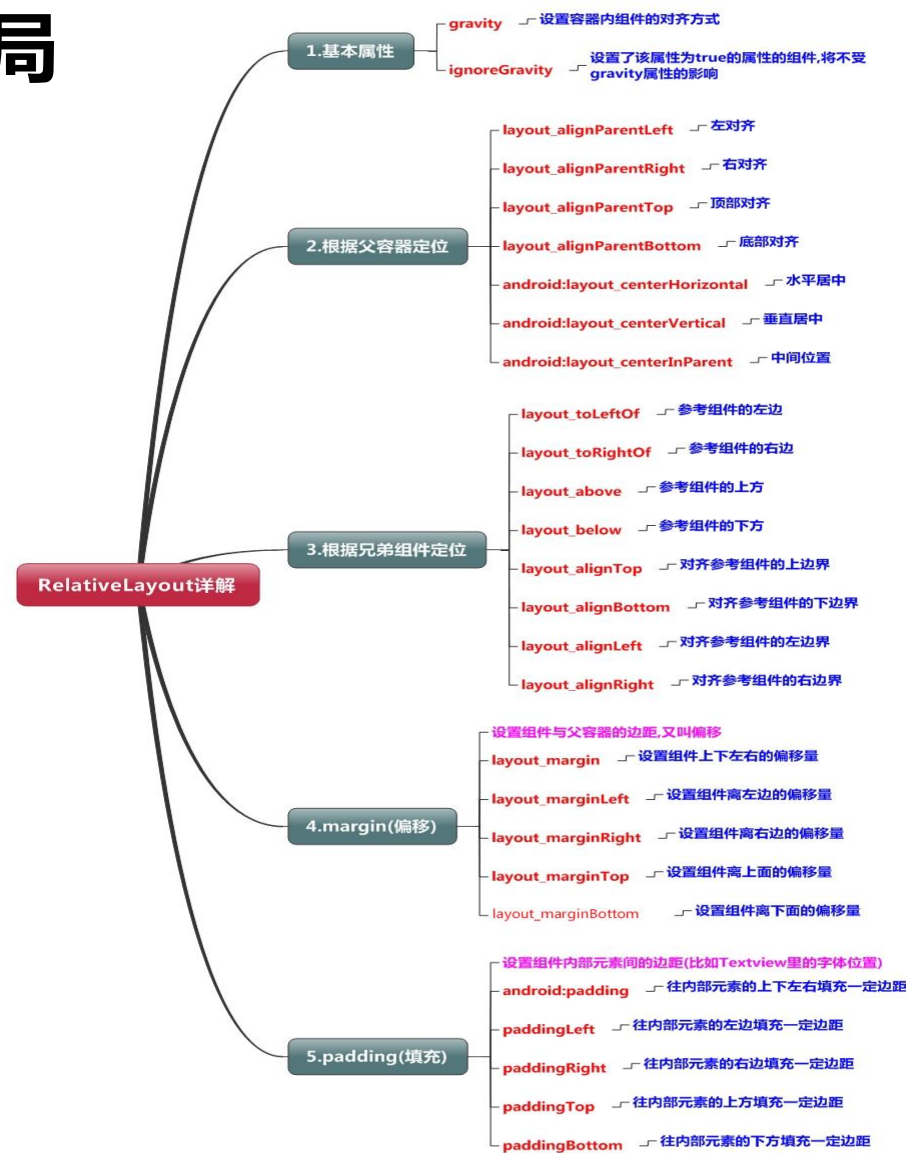
2.3 Android布局管理



常见布局

• 相对布局RelativeLayout

允许通过指定View对象相对于其他显示对象或父级对象的相对位置来布局。



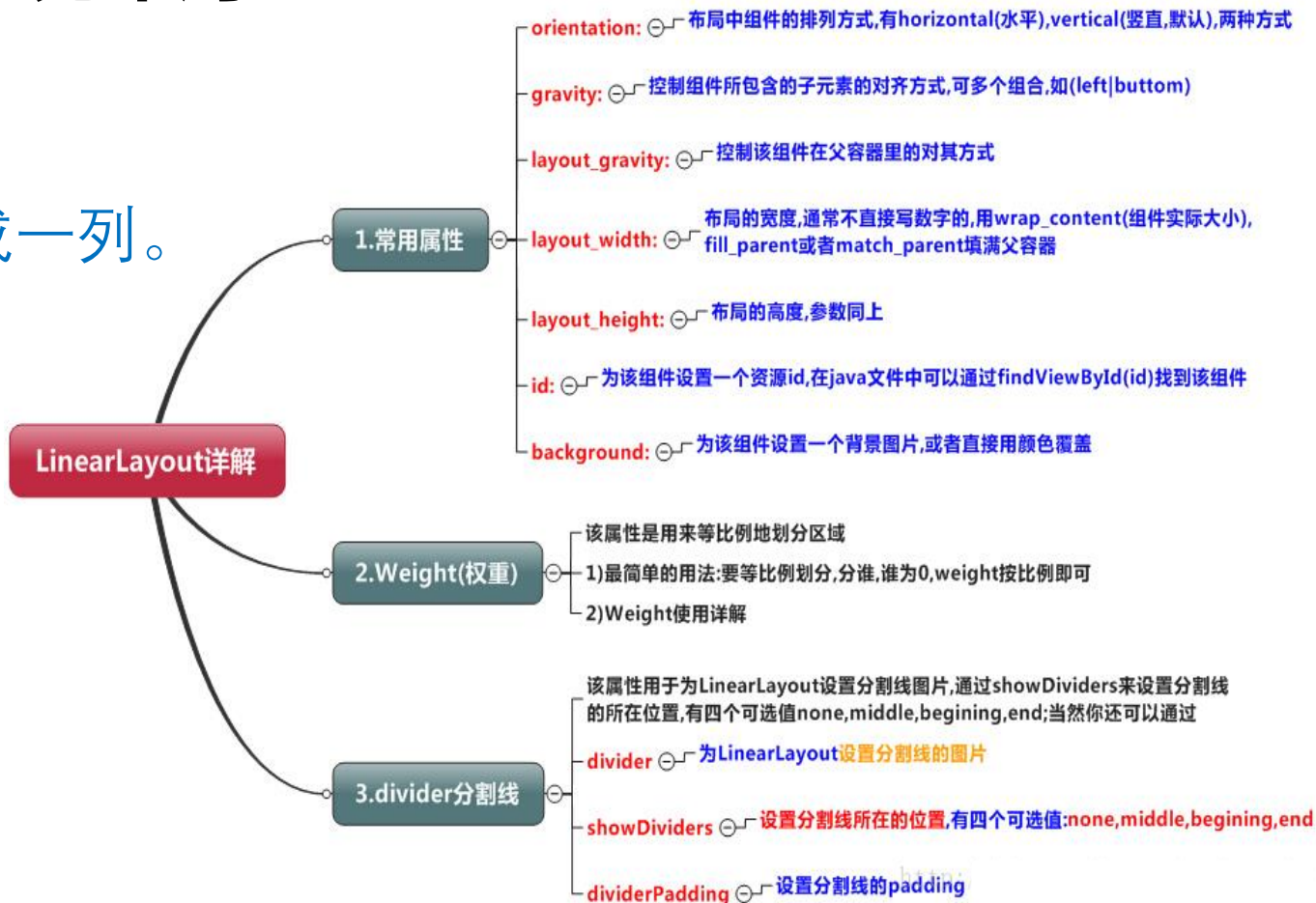
常见布局

- 线性布局LinearLayout

单一方向显示View对象，一行或一列。



线性布局

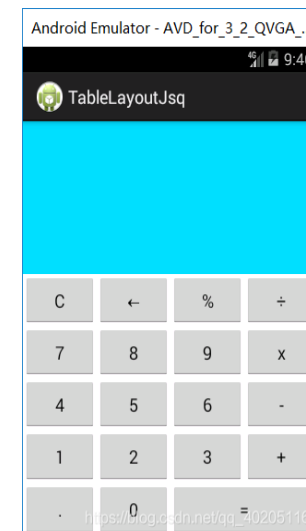


2.3 Android布局管理

常见布局

- 表格布局TableLayout

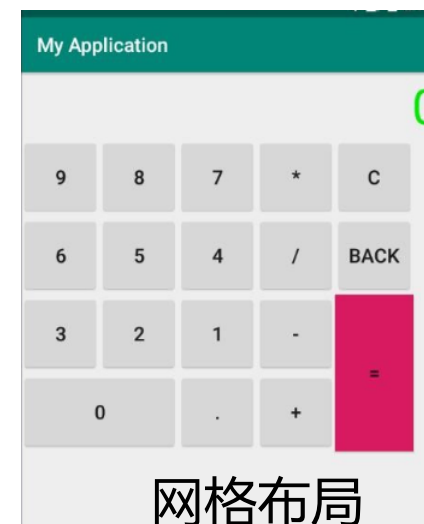
TableLayout以拥有任意行、列的表格对View对象进行布局，每个单元格内只显示一个View对象。TableLayout中的组件可以跨列，不能跨行。



表格布局

- 网格布局GridLayout

把整个容器划分成 n 行 \times m 列 的网格，行、列的序号都是从0开始。每个网格可以放置一个组件。和TableLayout不同的是，GridLayout容器中的控件可以跨多行也可以跨多列。



网格布局

常见布局

- 网格布局GridLayout



GridLayout相关属性图

常见布局

- 约束布局ConstraintLayout (Android**默认布局**)

使用扁平视图层次结构，无嵌套，根据同级视图和父布局的**约束条件**为每个视图定义位置。属性丰富，可创建复杂的大型布局，解决了复杂布局时嵌套过多的问题。

约束布局

- 约束布局ConstraintLayout是采用相对其它组件的位置的布局方式，通过指定ID关联其他组件，与之左右对齐、上下对齐或屏幕中央等方式来排列组件。

位置约束

约束布局采用方向约束的方式对控件进行定位，至少要保证水平和垂直方向都至少有一个约束才能确定空间的位置。

用法: **app:***param_name*= '@id/id_name'

- `layout_constraintLeft_toLeftOf`
- `layout_constraintLeft_toRightOf`
- `layout_constraintRight_toLeftOf`
- `layout_constraintRight_toRightOf`
- `layout_constraintTop_toTopOf`
- `layout_constraintTop_toBottomOf`
- `layout_constraintBottom_toTopOf`
- `layout_constraintBottom_toBottomOf`
- `layout_constraintBaseline_toBaselineOf`
- `layout_constraintStart_toEndOf`
- `layout_constraintStart_toStartOf`
- `layout_constraintEnd_toStartOf`
- `layout_constraintEnd_toEndOf`

约束布局

位置约束

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/str"
        android:id="@+id/textView"
        android:textColor="#CF352E"
        android:textSize="24sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```



约束布局

边距约束

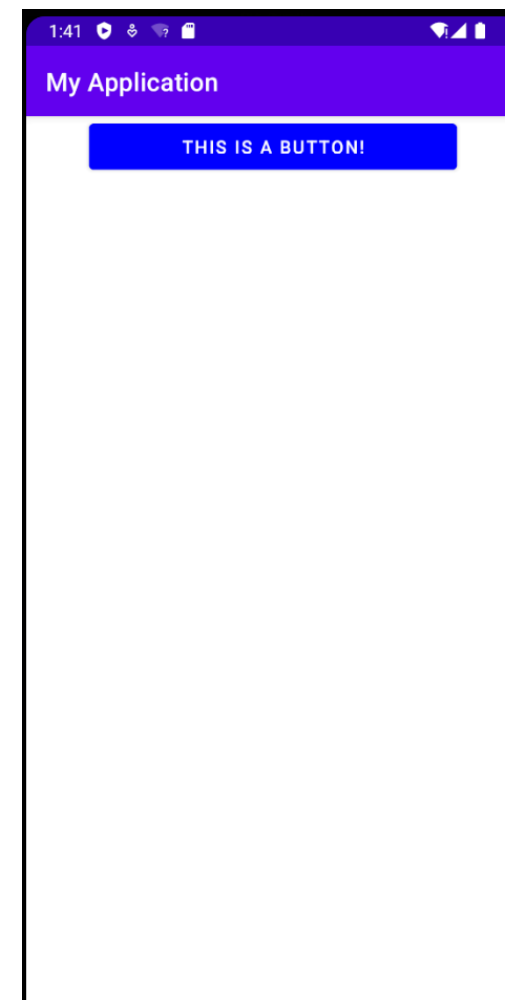
用法: **android:***param_name*= 'xxdp'

- `android:layout_marginStart`
- `android:layout_marginEnd`
- `android:layout_marginLeft`
- `android:layout_marginTop`
- `android:layout_marginRight`
- `android:layout_marginBottom`
- `layout_marginBaseline`

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button android:id="@+id/btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a Button!"
        android:backgroundTint="@color/blue"
        android:layout_marginLeft="50dp"
        android:layout_marginRight="50dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

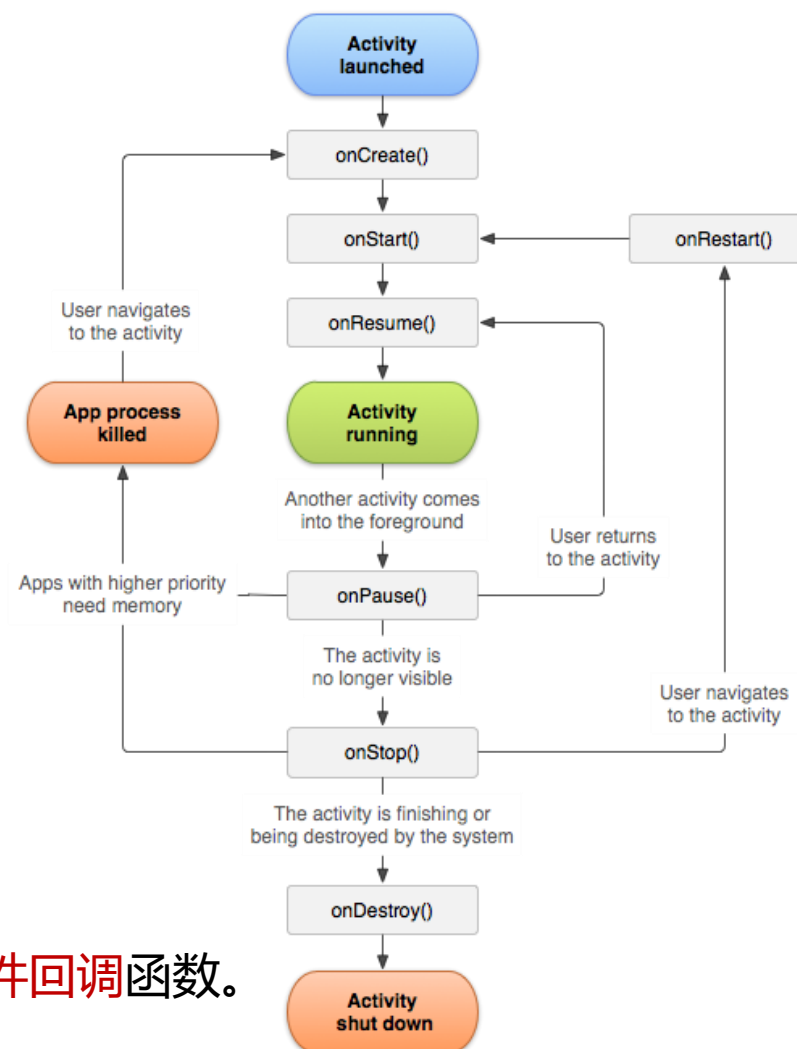


2.4 Activity生命周期

Activity：定义页面的显示时机，以及显示、退出时Activity状态信息的保存与恢复，用户交互操作时各控件的事件响应等控制逻辑。

- **启动状态**：Activity被压入栈顶。
- **活动状态**：Activity启动并获得焦点，可以与用户进行交互。一般，Activity启动后即处于运行状态。
- **暂停状态**：Activity失去焦点，不能与用户交互，但依然可见。
- **停止状态**：Activity不可见。
- **销毁状态**：更高优先级的应用需要内存时，系统将处于暂停或者停止状态的Activity从内存中删除，此Activity处于销毁状态。

随着Activity自身状态的变化，Android系统会自动调用不同的事件回调函数。



2.5 Android事件处理机制



Android事件

点击“单机游戏”

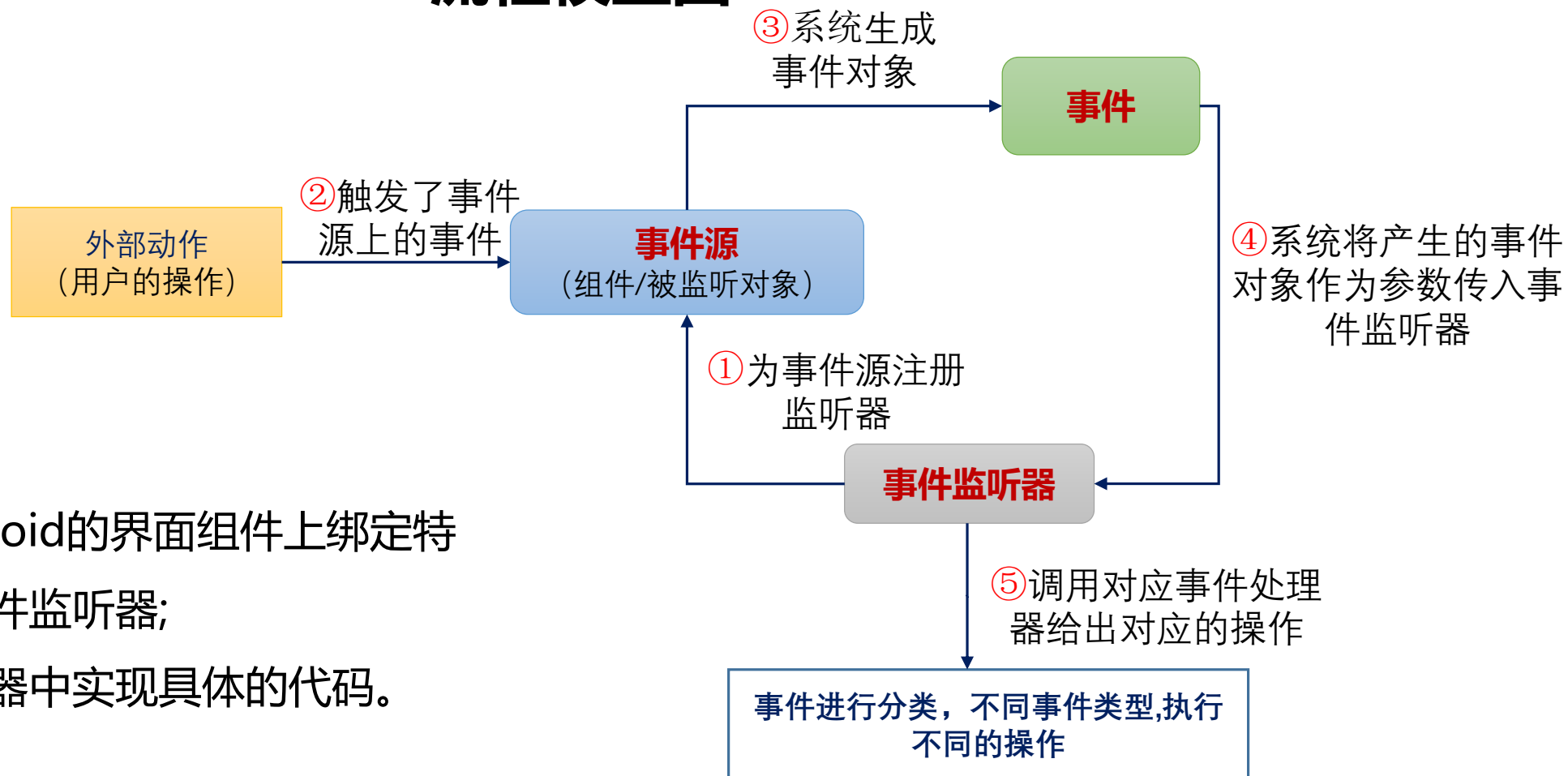


Android的两种事件处理模型

- 基于监听的事件处理;
- 基于回调的事件处理。

2.5 Android事件处理机制

流程模型图



总结：

- ① 在Android的界面组件上绑定特定的事件监听器;
- ② 在监听器中实现具体的代码。

基于监听的事件处理方法的实现方式

- 在布局文件中绑定
- 直接使用Activity作为事件监听器
- 内部类实现方式
- 匿名内部类实现方式
- 外部类实现方式

基于监听的事件处理方法的实现方式

1. 在布局文件中绑定

① 设置控件属性

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="按钮"  
    android:onClick="myclick"/>
```

② 在Activity中实现自定义方法

```
package com.jay.example.caller;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Toast;  
  
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
    //自定义一个方法,传入一个view组件作为参数  
    public void myclick(View source)  
    {  
        Toast.makeText(getApplicationContext(), "按钮被点击了", Toast.LENGTH_SHORT).show();  
    }  
}
```

基于监听的事件处理方法的实现方式

2. Activity作为事件监听器

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener
```

1

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
```

```
    Button btn1 = (Button) findViewById(R.id.myButton1);
```

```
    Button btn2 = (Button) findViewById(R.id.myButton2);
```

```
    btn1.setOnClickListener(this);
```

```
    btn2.setOnClickListener(this);
```

```
}
```

```
@Override
```

```
public void onClick(View v){
```

```
    switch (v.getId()){
```

```
        case R.id.myButton1:
```

```
            //执行某些操作
```

```
            break;
```

```
        case R.id.myButton2:
```

```
            //执行某些操作
```

```
            break;
```

```
    }
```

```
}
```

2

3

① Activity实现处理单击事件的接口

② 为事件源注册事件监听器，参数是this。

③ 重写事件处理方法，将事件源作为参数传入方法。

使用switch判断按钮的id在新版本中会出现以下报错，请改成if-else.

⚠ Resource IDs will be non-final by default in Android Gradle Plugin version 8.0, avoid using them in switch case statements :36

⚠ Resource IDs will be non-final by default in Android Gradle Plugin version 8.0, avoid using them in switch case statements :41

基于监听的事件处理方法的实现方式

3. 内部类

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Button btn1 = (Button) findViewById(R.id.myButton1);  
        Button btn2 = (Button) findViewById(R.id.myButton2);  
  
        btn1.setOnClickListener(new ClickEvent())  
        btn2.setOnClickListener(new ClickEvent());  
    }  
}
```

1

2

```
class ClickEvent implements View.OnClickListener{  
    public void onClick(View v){  
        switch (v.getId()){  
            case R.id.myButton1:  
                //执行某些操作  
                break;  
            case R.id.myButton2:  
                //执行某些操作  
                break;  
        }  
    }  
}
```

3

- ① 为事件源注册事件监听器，参数是内部类的对象。
- ② 在内部类中实现单击处理事件的接口。
- ③ 重写事件处理方法，将事件源作为参数传入方法。

基于监听的事件处理方法的实现方式

4. 匿名内部类

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Button btn1 = (Button) findViewById(R.id.myButton1);  
        Button btn2 = (Button) findViewById(R.id.myButton2);  
  
        btn1.setOnClickListener(1 new OnClickListener(){  
            @Override  
            public void onClick(View v){  
                //执行某些操作  
            }  
        });  
  
        btn2.setOnClickListener(new OnClickListener(){  
            @Override  
            public void onClick(View v){  
                //执行某些操作  
            }  
        });  
    }  
}
```

① 为事件源注册事件监听器，参数是匿名内部类的对象。

② 在匿名内部类中实现单击处理事件。

基于监听的事件处理方法的实现方式

5. 外部类

① 新建一个外部类，实现事件处理接口，自定义事件处理方法。

```
package com.jay.example.innerlisten;

import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;

public class MyClick implements OnClickListener {
    private TextView textshow;
    //把文本框作为参数传入
    public MyClick(TextView txt)
    {
        textshow = txt;
    }
    @Override
    public void onClick(View v) {
        //点击后设置文本框显示的文字
        textshow.setText("点击了按钮!");
    }
}
```

② Activity中创建外部类对象，将事件源作为参数传入。

```
public class MainActivity extends Activity {
    private Button btnshow;
    private TextView txtshow;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnshow = (Button) findViewById(R.id.btnshow);
        txtshow = (TextView) findViewById(R.id.textshow);
        //直接new一个外部类，并把TextView作为参数传入
        btnshow.setOnClickListener(new MyClick(txtshow));
    }
}
```

基于回调的事件处理方法

- 监听事件处理时事件源与事件监听器分开的，而基于回调的事件处理中UI组件不但 是事件源，而且还是事件监听器，通过组件的相关回调方法处理对应的事件。
- 为了实现回调机制的事件处理，Android为所有的GUI组件都提供了回调方法。

开发流程

- 自定义View类。
- 重写回调方法。

- *boolean onKeyDown(int keyCode , KeyEvent event);用户在该组件上按下时触发的
- *boolean onKeyLongPress(int keyCode ,keyEvent event);用户在该组件上长按时触发的
- *boolean onkeyShortcut(int keyCode ,keyEvent event);当一个键盘快捷键事件发生时触发该方法
- *boolean onKeyUp(int keyCode , keyEvent event);用户松开按键的时候触发的事件
- *boolean onTouchEvent(MotionEvent event);用户在该组件上触发触摸屏时触发该方法
- *boolean onTrackballEvent(MotionEvent event);用户在该组件商触发轨迹球时触发该方法

2.6 Android页面切换与数据传递

Intent

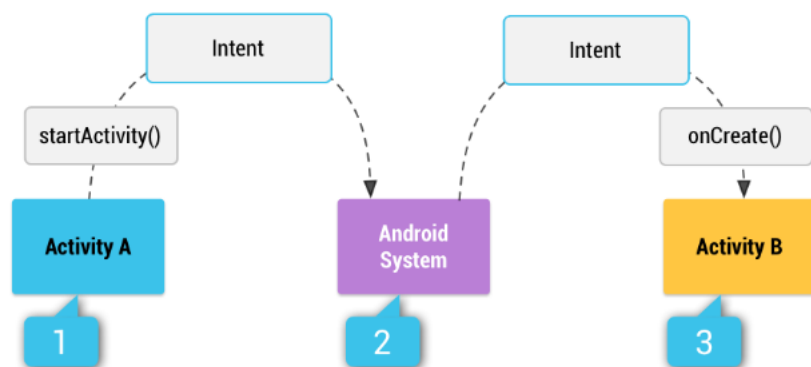
Intent 是一个消息传递对象，在应用程序运行时连接两个不同组件。

主要功能：

- 启动Activity
- 启动Service
- 启动广播

2.6 Android页面切换与数据传递

使用Intent启动Activity



ActivityA.java

```
Intent intent = new Intent( packageContext: ActivityA.this, ActivityB.class);  
intent.putExtra( name: "user", value: "test");  
startActivity(intent);
```

ActivityB.java

```
String str = getIntent().getStringExtra("user");
```

1. Activity A创建Intent, 指定要启动的Activity名字ActivityB, 如需传递数据可调用Intent对象的putExtra()方法。
2. Activity A调用startActivity方法, 将intent作为参数传入;
3. Android系统自动调用Activity B的onCreate()方法, 以此启动新的Activity。

2-1 游戏首页

- 设计飞机大战APP的游戏首页。

2-2 单机模式选择页面

- 设计单机模式选择界面；
- 监听游戏首页中“开始游戏”按钮的点击事件，事件发生时使用Intent实现游戏首页到单机模式选择页面的跳转，同时将音乐是否开启作为Intent的参数进行传递。

2-1 游戏首页

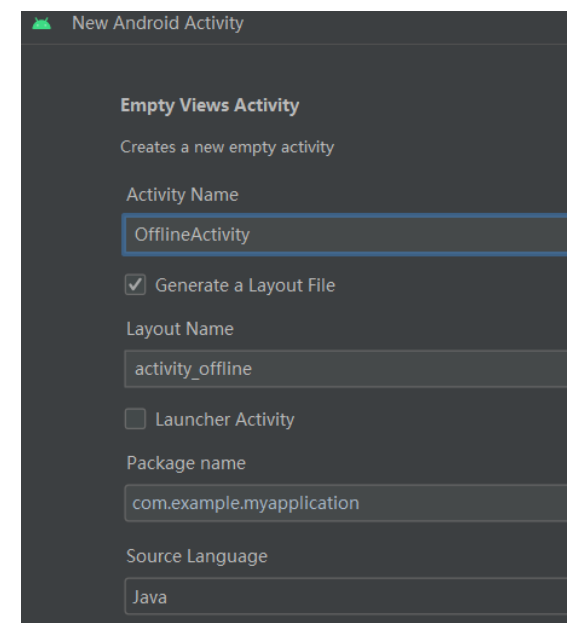
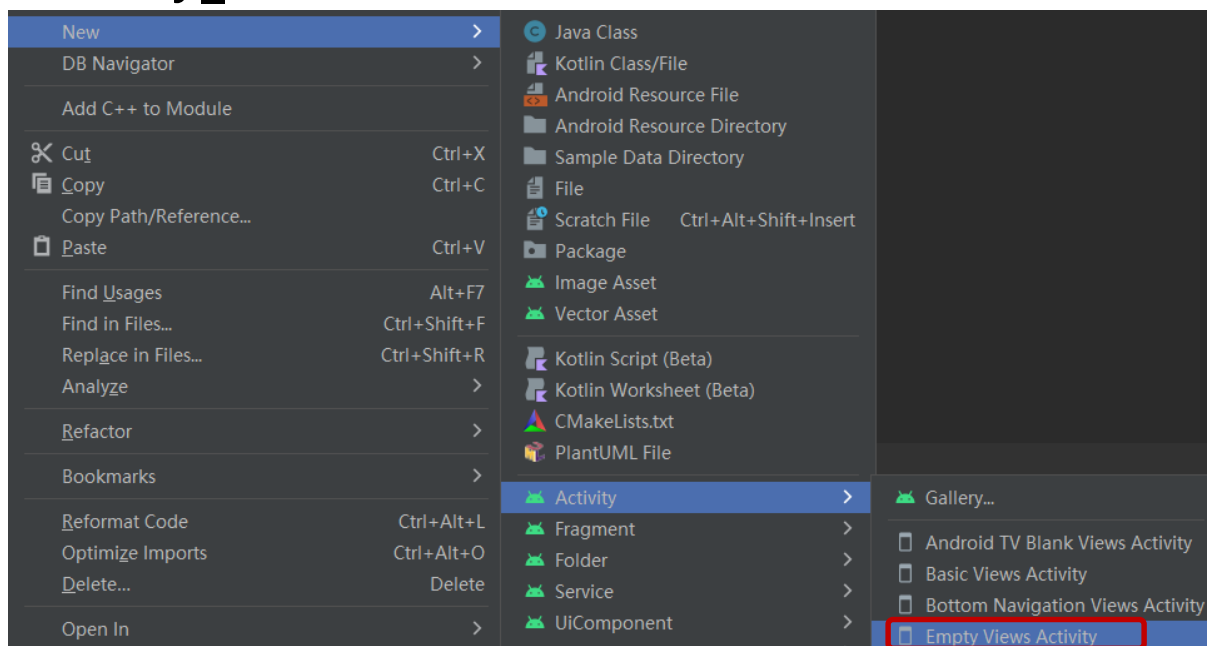
在实验一创建的 “AircarftWar2024” 的项目中，修改
res/layout/activity_main.xml:

1. 将bg_start.jpg（或者使用自定义图片）放到res/drawable文件夹下；
2. 添加一个全屏的ImageView控件，设置背景图片为bg_start.jpg，
3. 添加“ 单机游戏” 按钮；
4. 添加一组单选按钮，名为“开启音乐” 和 “关闭音乐”，默认选中“关闭音乐”；



2-2 单机游戏模式选择页面

1. 右击com.example.AircraftWar2024文件夹，创建一个新Activity，类型为“Empty Views Activity”，类名为OfflineActivity，语言选择JAVA，创建成功后会自动在AndroidManifest文件中注册OfflineActivity，同时在res/layout文件夹下会自动生成activity_offline.xml布局文件。



2-2 单机游戏模式选择页面

2. 在页面上添加一个文本框，内容为“单机模式难度选择”；
3. 在页面上添加三个按钮，“简单模式”、“普通模式”和“困难模式”；
4. 在游戏首页点击“开始游戏”，实现从游戏首页到单机游戏模式的跳转，同时将音乐开关是否开启作为Intent参数进行传递。

